Computer Science

COMPSCI 105 SS – Assignment Two

January 2007

Electronic Submission Due: 4:00 pm Thursday 25th January

Worth 10% of your final mark

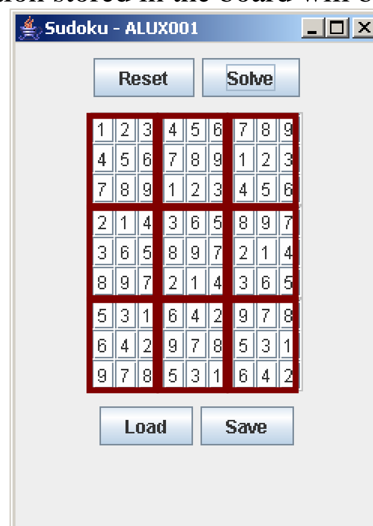
Aims of the assignment

Specific objectives are listed below:

- Use the Java IO system to read and write text files
- Use try catch blocks to handle exceptions correctly
- Use recursion to solve problems
- Use sorting algorithms to sort information.

Part one: Sudoku I/O

For this part, you will have to extend the Sudoku application you worked on during Assignment 01. For this section, you should add functions that allow the user to load or save the game. The information stored in the board will be saved as plain text.



A partially completed version of the Sudoku application is available on the 105 web site. You will need to implement the code in the SudokuIO class. The documentation in the SudokuIO.java skeleton file explains which methods you have to create and what those methods should do.

When the user clicks on the Load or Save button, a dialog box should appear that allows the user to enter a file name for loading or saving the board. The board should be saved as a single line of text, 81 characters in length. Each character represents a position on the board, with the first 9 characters representing the first row of the board, then each subsequent set of 9 characters representing a subsequent row in the board. An empty cell should be represented by a full stop.

You will need to use a JFileChooser to create the dialog box. See the Java API for details and example of how to use JFileChooser.

Part two: Reading, sorting and writing text files

Write a program called "Extract" that accepts two command line arguments. The first argument is the name of a text file used as input. The second argument is the name of a file that will be used for output from the program.

Your program should read the text from the input file and break the text up into individual words. The words should all be converted into lower case, and should be sorted into alphabetical order. Any duplicate words should be discarded and the list of unique words should be written to the output file. The program should print out a message showing your UPI, the name of the input file and the number of unique words written to the output file.

For example, if the application was called as follows:

```
java Extract input.txt output.txt
```

and the file input.txt contained the text:

```
This is
a small file.
```

then the file output.txt would contain the text:

```
a
file
is
small
this
```

and the following message would be printed to standard output:

```
ALUX001:  input.txt contains 5 unique words
```

Note that any characters that are not letters should be treated as spaces for the purposes of this program. For example, an input file that contains the text:

```
She said, “that’s mine, not yours!”
```

should result in the output

```
mine
not
s
said
she
that
yours
```

You may find the `String.split()` method helpful for this question.

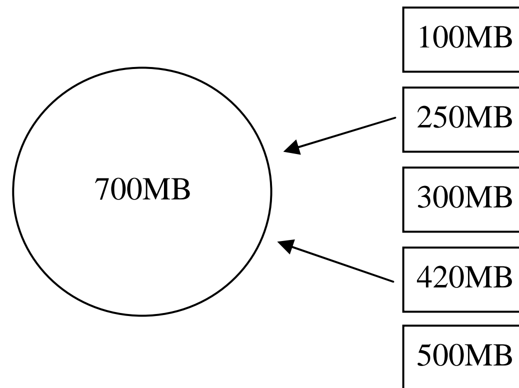
Part three: Recursion

For this question you need write a program called `CDWaste` which calculates the minimum amount of wasted, or left over, space there will be given a CD of a specified size and a collection of files of different sizes to be written to the CD.

For example, assume that you have a collection of 5 files with sizes:

100MB, 250MB, 300MB, 420MB and 500MB

and that you have a CD with a 700MB capacity.



In order to minimize the amount of wasted space, the 250MB file and the 420MB file should be written to the CD. The amount of wasted space in this case will be:

700 – (250 + 420) = 30MB.

No other combination of files will result in a smaller amount of wasted space.

The input to your program must come from the command line, where the first argument specifies the total size of the CD, and the remaining arguments list the individual file sizes. For example, the minimum amount of wasted space in the above situation could be calculated using:

```
c:\105\Asst2> java CDWaste 700 100 250 300 420 500
Minimum waste by abcd001: 30
```

Notice that the output must include your UPI. You can assume:

- the total size of the CD and the sizes of the files are positive integers
- no file can be written more than once to the CD (so for any file, it is either written or not written to the CD)
- there can be more than one file with the same size, and the file sizes may be specified in any order (ie. not necessarily increasing numerical order as in the example above)

Your program must define the following **recursive** method which will compute the minimum waste:

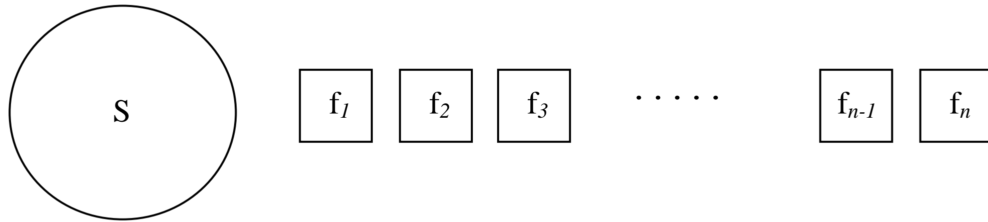
```
public static int minWaste(int spaceLeft, int[] fileSizes, int numFiles)
```

where:

- `spaceLeft` is the amount of space left on the CD
- `fileSizes` is an array storing all of the file sizes, and
- `numFiles` specifies the number of file sizes in the array being considered

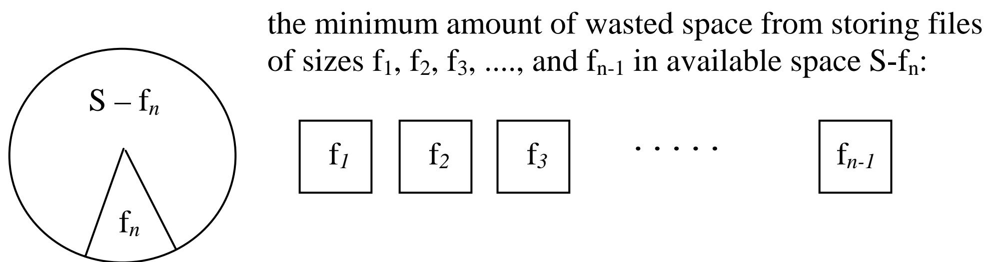
Think carefully about how this problem would be defined recursively. The following explanation may be useful:

If the amount of available space to store the files is S, and the files we are considering have sizes of f_1, f_2, f_3, ..., f_{n-1} and f_n:

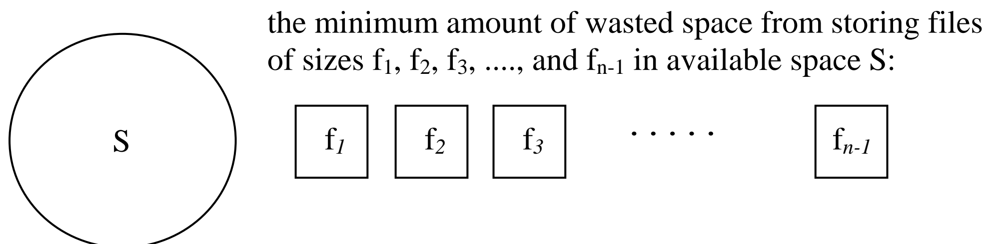


Then the overall minimum amount of wasted space will be the MINIMUM of the following two values **A** and **B**:

A) if we include f_n



B) if we don't include f_n



Another example of the program running (using a very small CD!) is shown below:

```
c:\105\Asst2> java CDWaste 30 4 6 12 13 18 21
Minimum waste by abcd001: 0
```

In this case, the minimum amount of wasted space is 0, because 12 + 18 = 30.

NOTE:

- Your UPI must appear in the output, as shown in the example above

Part four: Sudoku Solver (optional)

This section is optional and contributes no marks. However, it will be good practice with recursion and will be satisfying to complete :)

In order to solve the board, you should try to think about the following questions:

- How can you define the problem in terms of a smaller problem of the same type?
- How does each recursive call diminish the size of the problem?
- What instance of the problem can serve as the base case?
- As the problem size diminishes, will you reach this base case?

The SudokuAI class should contain all the code required to perform the "Solve" function. The documentation included in the SudokuAI.java file will provide some hints about how to approach this problem.

You do not need to alter any other classes or methods belonging to the Sudoku application. You may find the SudokuBoard method copy() to be useful in solving this problem.

Marking Schedule
Files Required: `Extract.java, CDWaste.java, SudokuIO.java` Optional files: `SudokuAI.java`

SudokuIO.java	
The board is saved correctly when the save button is pressed	5
The board is loaded correctly when the load button is pressed	5
Extract.java	
The application reads the input file and writes the contents of the file in sorted order to the output file	4
Efficient algorithms are used throughout the code	2
Only letters are included in the output, no non-letter characters	2
The application correctly calculates and prints the number of unique words	2
CDWaste.java	
The `minWaste()` method is recursive, and structured correctly with a base case and recursive calls	2
The correct amount of wasted space is calculated correctly in all cases	8

Style of the algorithms	5
Neatness and consistency of the code	5

Total	40